Business Process Integration by using General Rule Markup Language

Milan Milanović¹, Nima Kaviani², Dragan Gašević³, Adrian Giurca⁴, Gerd Wagner⁴, Vladan Devedžić¹, Marek Hatala²
¹FON-School of Business Administration, University of Belgrade, Serbia
²Simon Fraser University Surrey, Canada
³Athabasca University, Canada
⁴Brandenburg University of Technology at Cottbus, Germany
milan@milanovic.org, nkaviani@sfu.ca, dgasevic@acm.org, giurca@tu-cottbus.de, wagnerg@tu-cottbus.de, devedzic@etf.bg.ac.yu, mhatala@sfu.ca

Abstract

A business process usually includes multiple business partners that use systems with their business logics represented in different rule (or policy) languages. The integration of business processes is a major goal followed by business enterprises and involves the interchange of business rules and policies between partners. However, the variety of business rules employed by the partners presents a significant burden to the integration of collaborating business systems. In this paper, we propose a solution that enables translating rules and policies defined in different rule languages into a single general rule language (REWVERSE II Rule Markup Language - R2ML) and processing them in a uniform manner. This provides a unified view of different partners' business rules in an integration process, while permitting the partners to continue to leverage their own business rules without changing their technologies. We show how the concepts of the KAoS policy language can be transformed to R2ML and then from R2ML to the other rule (or policy) languages.

1. Introduction

Allowing heterogeneous systems to interoperate is one of the major challenges in IT research. The problem of how to provide transparent access to heterogeneous information sources while maintaining their autonomy has been recognized a while ago. The key challenge in cases involving multiple organizations (and systems) is that some rules that support business policies might directly not be executable by another side of a process.

The integration process between parties involved in transactions or procedures (e.g., credit card authoriza-
rules, which have a higher level of abstraction, [2] [22], and that we should be able to share those rules by using a unique representation, i.e., by using a general rule language [10]. This issue has been recognized by W3C and manifested in the Rule Interchange Format (RIF) initiative [3], which tries to define a standard for sharing rules. That is, RIF should be expressive enough, so that it can represent concepts of various rule languages. Besides requirements it defines, it also proposes a number of use-cases in which RIF could be used. Such a rule language should enable modeling different types of rules such as derivation rules considering the importance of inferring new facts and integrity rules considering the deontic, i.e., “must”, nature of policy languages [21].

In our approach, we propose the use of REWERSE Rule Markup Language (R2ML) [21], which addresses almost all use-cases and requirements for RIF) [3], along with a set of transformations between rule and trust management policy languages. We illustrate the benefits of our approach by using R2ML as a means to share policies in the process of trust and transaction negotiation. In the next section, we motivate our research by describing an example based on one of the RIF use cases identified by RIF Working Group.

2. Motivation

Today, the integration process between parties is usually implemented by using (Semantic) Web services. Web services are of the most important domains for applying Web policy rules. Policies can be regarded as constraints to be combined with Web services to explicitly identify conditions under which the use of a service is permitted or prohibited [7]. However, due to the diversity of existing policy languages, chances are that the policies used to protect the services or resources are not defined in the same language, which makes the process of communication impossible.

As an example, let us consider an eCommerce scenario proposed in RIF use cases requirements working draft [3]. The example considers a buyer and seller, with two different policy languages, who need to negotiate their policies and preferences. In this scenario, a buyer wants to buy a device from an online web shop (shown in Figure 1). The buyer employs software called BuyerAgent that functions as the automated negotiating agent, while the web shop server employs another software called SellerAgent as the automated negotiating agent. The policies for the buyer and the seller describe who they trust, and what purposes for. The negotiation is based on these policies specified as rules and credentials for SellerAgent and BuyerAgent.

The policies can be interchanged (i.e., negotiated and disclosed) in this process to establish the trust between the involved parties. Assuming that BuyerAgent and SellerAgent use different languages for defining and evaluating rules (policies), e.g., KAoS by the BuyerAgent and F-Logic rules by the SellerAgent, an interchange language needs to be employed for enabling the rules to be transformed between the two software systems and to facilitate the process of negotiation.

![Figure 1. eCommerce scenario: negotiating transactions through disclosure of Buyer and Seller policies](image)

Referring to Figure 1, when the buyer wants to buy a product and clicks on the «buy it» button at the online web shop (step 1), its BuyerAgent takes over and sends a request to the online web shop (step 2). SellerAgent receives the request, retrieves its policies stored in its rule repository (step 3 and 4) and sends parts of its policy as sets of rules back to the BuyerAgent (step 5). The policy states, “If a buyer provides the delivery information (address, postal code, city, and country) then he must provide his credit card as well”. SellerAgent stores its policies in terms of F-logic rules in its rule repository. The rule for the policy above has been shown in Figure 2.

![Figure 2. A sample of F-logic rule](image)

Establishing an agreement between BuyerAgent and SellerAgent about the language in which they share their rules is the ideal case that seems to be impossible due to the variety of resources, broker agents, and Web rule languages that they may use. An easier way is to send the rules (policies) in the language in which they
have been originally defined (i.e., F-Logic for the SellerAgent in our scenario). In case that there is no understanding about F-Logic by BuyerAgent or no knowledge about KaoS by the service provider, the communication would fail. So, either BuyerAgent or SellerAgent must be able to translate the policy to an equivalent KaoS policy (rule). By using R2ML as the interchange language, SellerAgent does not send the policy in the F-Logic format, but it translates it to the R2ML interchange format and sends it out (step 5 in Figure 1). BuyerAgent retrieves and translates the R2ML rule to the format it uses, i.e., KaoS (step 6 in Figure 1). The KaoS policy result of this transformation can be similar to the policy defined in Figure 3.

After retrieving this policy, BuyerAgent and SellerAgent interchange some more messages to identify the level of trust in each other. Once the desired level of trust is established and BuyerAgent sends its delivery information, it also sends the credit card information, again in the format of R2ML (step 7 in Figure 1). SellerAgent retrieves the information and translates it from the R2ML format into the F-logic format (step 8 in Figure 1). Thereafter, SellerAgent sends a message to BuyerAgent confirming the completion of the purchase (step 9 in Figure 1).

In this and similar eCommerce negotiation processes, the buyer agents and the seller agents must have different transformers from each policy (rule) language to the others. This is hard to achieve due to the vast variety of these languages. Considering the number of available policy and rule languages, it would be a lot of effort to develop one-to-one transformations between the policy languages. For \( n \) policy languages, we would need \( n*(n-1) \) transformations. It gets even worse if we consider the constant changes that should be applied to the transformations due to the improvements and extensions of each policy language. On the contrary, using an intermediary language would considerably reduce the number of transformations. For \( n \) policy languages, we will need \( n \) transformations to the intermediary policy language, and then \( n \) transformations from this language back to all the policy languages. Thus, the number of transformations will boil down to \( 2*n \). Furthermore, the process of transformation is done by the experts who share a common knowledge in the domain of R2ML and other policy languages. This hides the complexity of understanding multiple languages from other domain users, policy engineers, and system analyzers and guarantees sound mappings between different rule (policy) languages.

Figure 3. An excerpt of an equivalent KaoS policy of the policy shown in Figure 2

In order to address the problem, the focus should be on finding an interchange format language with the highest possible degree of interoperability to cope with different available policy and rule languages. Addressing the problem of negotiation in scenarios similar to the one explained above would significantly help to enable inter-party interactions.

In the rest of the paper, we propose a new solution to interchange the policy and rule languages from one to another using a general rule markup language (R2ML), with as less information loss as possible.

3. Background

In this section, we describe some of the most known policy and rule languages by their corresponding communities.
3.1 Policy languages

As mentioned earlier, there are several policy languages proposed so far, with the goal of protecting the privacy of information and authorizing requesters by providing different levels of access to the available resources and information. The syntax of these languages varies from rigid ordinary logic languages such as Cassandra [1], which is based on Constraint Language Programming (CLP), PeerTrust [16] and PRO-TUNE [2], which use a Prolog meta-interpreter, to more relaxed markup languages such as KAoS [20] and Rei [8]. Discussing all of the available policy languages is beyond the purpose of this paper and here we just review the characteristics of some of the most recent policy languages.

**PeerTrust** is a trust negotiation engine for the Semantic Web and P2P networks [16]. PeerTrust’s language is based on first order Horn rules (definite Horn clauses), i.e., rules of the form:

\[ \text{lit}_1 \leftarrow \text{lit}_i (i=1..n) \]

where each \( \text{lit} \) is a positive literal of the form \( P(t_1, ..., t_n) \), where \( P \) is a predicate symbol, and \( t_i \) \( (i=1..n) \) are the arguments of this predicate.

**Cassandra** is another policy language based on CLP [1]. It uses a policy language based on Datalog constraints and its expressiveness can be adjusted by changing the domain of the constraints. Policies are specified using the following predicates which govern access control decisions: \( \text{permits}(e, a) \) specifies who can perform which action; \( \text{canActivate}(e, r) \) defines who can activate which role \( e \) is a member of \( r \); \( \text{hasActivated}(e, r) \) defines who is active in which role; \( \text{canDeactivate}(e, r) \) specifies who can revoke which role; and \( \text{isDeactivated}(e, r) \) is used to define automatically triggered role revocation.

**Rei** language is a part of the Rei policy framework that permits specification, analysis and reasoning about declarative policies defined as norms of behavior [8]. Rei adopts a rule-based approach to specify semantic policies. Rei policies restrict domain actions that an entity can/must perform on resources in the environment, allowing policies to be developed as contextually constrained deontic concepts, that is, permission, prohibition, obligation and dispensation. The current version of Rei (2.0) adopts OWL-Lite to specify policies and can reason over any domain knowledge expressed in either RDF or OWL.

**KAoS** policy language is a part of the KAoS framework that provides policy and domain management services for agent and other distributed computing platforms [20]. It has been deployed in a wide variety of multi-agent and distributed computing applications. KAoS policy services allow for the specification, management, conflict resolution, and enforcement of policies within agent domains. The KAoS language adopts an ontology-based approach for semantic policy specification. In fact, policies are mainly represented in OWL as ontologies, which make it possible to combine them with Semantic Web Services and then use them to define the policies of a Web service provider. Defining KAoS policies as OWL expressions gives the language more flexibility to maneuver over the concepts of the domain. Different quantifying expressions, inheritance relationships, cardinality restrictions, etc. can be expressed explicitly in KAoS thanks to the constructs of OWL. Exploiting the OWL to define the policies also enables KAoS to perform static conflict resolution and policy disclosure. KAoS has its classes and properties already defined in OWL ontologies, referred to as KAoS Policy Ontologies (KPO) [20]. A KAoS policy has properties to control or oblige the performance of an action by dictating a set of restrictions to the properties of an action. The control over an action can be further extended by placing constraints on the current state or the history of the events in a domain by defining a requiresConditions element. Furthermore, the order of executing an obliged action can be controlled by defining obligationConstraints that regulate the enforcement process (see Figure 3).

3.2 Rule languages

Currently, there is no standard for defining and sharing rules on the Web. The most important initiative is called Rule Interchange Format (RIF) [3], which defines a set of requirements and use cases for sharing rules on the Web. It is important to point out that the purpose of this language is to serve as an intermediary language between various rule languages. It should not provide a formally-defined semantic foundation for reasoning on the Web such as OWL for ontologies. Here we name three most prominent rule languages which can be used for integration with RIF and briefly describe their characteristics.

**RuleML** is a markup language for publishing and sharing rule bases on the World Wide Web [5]. RuleML builds a hierarchy of rule sublanguages upon XML, RDF, XSLT, and OWL. The current RuleML hierarchy consists of derivation (e.g., SWRL, FOL), integrity (e.g., OCL invariant), reaction (aka Event-Condition-Action - ECA), transformation (e.g., XSLT) and production rules (e.g., Jess). RuleML is based on Datalog and its rules are defined in the form of an implication between an antecedent and a consequent, with the meaning that whenever the logical expression in the antecedent holds, then the consequent must also hold. However, an important constraint of RuleML is that, it...
has a limited number of transformations implemented, which limits its proven usability in solving practical problems.

The Semantic Web Rule Language (SWRL) is a proposed rule language based on the W3C Web Ontology Language, OWL [6]. Similar to RuleML rules, a SWRL rule is also in the form of an implication and is considered another type of axiom on top of the other OWL axiom types. This means that SWRL rules are usually used for defining integrity constraints similar to OCL in UML, but they can be used for defining derivation rules as well. Both consequent and antecedent are collections (i.e., conjunctions) of atoms. It should be noted that the purpose of SWRL is not to be universal rule syntax for interchanging rules, as it cannot represent many linguistic constructs of other rule languages (e.g., F-Logic, KaoS, or OCL).

F-Logic is a deductive, object-oriented database language that combines the declarative semantics and expressiveness of deductive database languages with the rich data modeling capabilities supported by the object-oriented data model [11]. The basis for a logic programming language is given in [11]. There are many implementations of the language such as FLIP or, TRIPLE. In this paper, we refer to the syntactic and semantic capabilities of F-Logic as implemented by Ontoprise GmbH (http://www.ontoprise.de).

4. R2ML as a Rule Interchange Language

We propose the use of a general rule representation language with the syntactical capacity to represent various constructs for the policy and rule languages mentioned in the previous section. More specifically, we propose using the R2ML language [21]. R2ML is a general rule interchange language that possesses expressivity for representing four types of rules, namely, integrity, derivation, reaction, and production rules, trying to address all RIF requirements [3]. Besides rules, R2ML has its own set of constructs for representing vocabularies and domain ontologies similar to UML or OWL. Having in mind such an expressivity, we can use R2ML to represent the following artifacts related to the process of negotiation between parties discussed in the previous section:

- R2ML integrity and derivation rules can be used to represent trust management policy rules such as the ones defined in Rei, PeerTrust, PROTUNE and KaoS. R2ML integrity rule is defined as a constraint assertion, which is a sentence in a logical language such as first-order predicate logic or OCL [15], while R2ML derivation rules consist of a set of "conditions" and a "conclusion".

- R2ML reaction rules consist of a mandatory triggering event expression, an optional condition, and a produced action or a post-condition (or both). There are two types of reaction rules: those that do not have a post-condition, which are the well-known Event-Condition-Action (ECA) rules, and those that do have a postcondition, which we call ECAP rules.

- R2ML production rules can be used to simulate events by externally asserting corresponding facts into the working memory, in this way production rules can implement reaction rules, and derivation rules of the form if-Condition-then-assert-Conclusion using the special action assert that changes the state of a production rule system by adding a new fact to the set of available facts. So, this type of rules can be used to translate rules between different rule and policy languages, which can be represented with R2ML derivation and reaction rules.

- R2ML vocabularies can be used for representing domain vocabularies expressed as UML class diagrams or OWL ontologies.

Along with R2ML, a set of bi-directional transformations between R2ML and all the languages discussed in Section 2 needs to be developed for our proposed solution. Given such a set of transformations, we can transform KaoS policies (via R2ML) into F-Logic, or a UML vocabulary into OWL ontology [14]. In the rest of this section, we give a brief overview of R2ML.

4.1 R2ML: A brief overview

R2ML is a general rule interchange language that tries to address all RIF requirements [3]. The R2ML current version is 0.4 [18] and its abstract syntax is defined with a metamodel by using OMG’s Meta-Object Facility (MOF). This means that the whole language definition is represented by using UML diagrams, as MOF uses UML’s graphical notation. The full description of R2ML in the form of UML class diagrams is given in [18], while more details about the language can be found in [21]. In Figure 4, we give an excerpt of the metamodel that defines derivation rules that we will use in the following sections.

![Figure 4. The UML representation of a R2ML derivation rule](Image)

A derivation rule has a set of conditions and a conclusion (see Figure 4) with the ordinary meaning that the conclusion can be derived whenever the conditions
hold. While the conditions of a derivation rule are instances of the AndOrNafNegFormula class, representing quantifier-free logical formulas with conjunction, disjunction and negation; conclusions are restricted to quantifier-free disjunctive normal forms without NAF (Negation as Failure, i.e., weak negation).

Conditions and conclusions are both defined by the use of Atoms that are the basic constituents of a formula in R2ML. For our transformation of policy languages, we use R2ML ReferencePropertyAtoms in the condition and R2ML ObjectDescriptionAtom in the conclusion of a derivation rule. A ReferencePropertyAtom (shown in Figure 5) associates object terms as “subjects” with other object terms as “objects”. An ObjectDescriptionAtom (shown in Figure 6) refers to a class as a base type and to zero or more classes as categories, and consists of a number of property/term pairs (attribute data term pairs and reference property object term pairs). Any instance of such atom refers to one particular object that is referenced by an objectID, if it is not anonymous and is bound to the object or data values that are assigned to its properties. An example of a derivation rule is:

Example 1. If reservation date of a rental is five days in advance of the rental start date then rental discount is 10.

Terms are the basic constituents of atoms. Similar to atoms and following RuleML, the R2ML language distinguishes between object terms, data terms and generic terms. An ObjectTerm is an ObjectVariable, anObjectName, a ReferencePropertyFunctionTerm, or an ObjectOperationTerm (see Figure 7).

Figure 5. R2ML ReferencePropertyAtom

Figure 6. R2ML ObjectDescriptionAtom

An ObjectOperationTerm is formed with the help of a contextArgument, a user-defined operation, and an ordered collection of arguments. The ReferencePropertyFunctionTerm corresponds to a functional association end (of a binary association) in a UML class model. ObjectNames in R2ML are the same artifacts like Object in UML. They also correspond to the Individual concept of OWL. Variables are provided in the form of ObjectVariable (i.e., variables that stand for objects), DataVariable (i.e., variables that stand for data literals), and GenericVariable (i.e., variables that do not have a type).

The concept of data value in R2ML is related to the RDF concept of data literal. Following OWL, R2ML distinguishes between plain and typed literals. A DataTerm is a DataVariable, a DataLiteral, or DataFunctionTerm, which can be of three different types: DataOperationTerm, AttributeFunctionTerm, and DatatypeFunctionTerm.

5. Mapping R2ML and Rule languages

In this section, we describe the general mapping principles from R2ML markup language to the F-Logic language. The XSLT implementations of our transformations are available in [19].

As stated in the previous section, an R2ML derivation rule has conditions and a conclusion. Rules have also an optional attribute r2ml:id which identifies a unique rule in a derivation rule set. Any R2ML derivation rule is mapped into an F-Logic rule in the following form:

FORALL VList1 At1 AND ... AND Atn <- EXISTS VList2 Formula,

where VList1 represents a list of all variables appearing in the conclusion of the rule; VList2 represents a list of all variables appearing in the condition of the rule, but not in the conclusion; At1 AND ... AND Atn represents the R2ML conclusion enumeration of atoms using the F-Logic keyword AND; and Formula represents the
A positive R2ML ReferencePropertyAtom maps into an F-Logic F-molecule, that is, s[p->o] where s is the content of the child role element subject of ReferencePropertyAtom; p is the name of the property i.e., the referenceProperty value l and o is the content of the child role element object of the ReferencePropertyAtom. A positive R2ML ObjectClassificationAtom is mapped into an F-Logic membership declaration, that is, t:C where t is the corresponding term (object name and/or object variable) and C is the value of R2ML classID attribute.

A positive R2ML ObjectDescriptionAtom is mapped into a composite F-Logic F-molecule, that is, s[p l->v l; ..., p n->v n] where s is the content of the element R2ML subject of the R2ML atom; p_i is the value of referenceProperty or attribute (N.B., F-Logic do not make distinction between data and objects); and v_i is the content of object or value elements of the R2ML atom.

The R2ML ObjectName construct is used to encode individuals (i.e., objects) and is mapped into the F-Logic construct. It consists of a mandatory attribute, objectID and from an optional R2ML Class that denotes the class membership of the object.

Since in F-logic there is no distinction between different types of data literals, R2ML data literals (i.e., TypedLiteral and PlainLiteral) are mapped into F-Logic constants using the value of the R2ML lexical-Value and R2ML languageTag (in the case of plain literal) attributes.

R2ML distinguishes between three types of variables. The R2ML Variable, ObjectVariable and DataVariable are mapped into the F-Logic concept of variable using the value of the R2ML name attribute into an F-logic variable with that name and adding the F-Logic prefix for variables (i.e., the “?” character).

Figure 8 shows an excerpt of the rule that we described in Figure 1, converted to its R2ML equivalent based on the transformation rules explained in this section. The conversion shows how an F-logic rule is represented as an R2ML derivation rule. Due to the space limits, we do not give the explanations for transforming...
the other parts of the rule.

As conceptual mappings described in this section are bidirectional, they are the same for the R2ML to F-logic transformation as well [19]. In the similar way as we provided conceptual mappings between F-logic and R2ML, we also defined mappings between R2ML and languages such as Jess, JBoss and Jena for derivation rules together with their implementations [19].

6. Mapping between R2ML and Policy languages

In the previous section, we have shown how we mapped F-logic [11] to R2ML [18] and way back. The last step in providing a powerful interchange format between different parties in business process integration based on the rule interchange is to map policy rules from a source policy (rule) language (e.g., KAoS [20]) to R2ML and vice versa.

In case of the RIF use case scenario shown in Figure 1, and the policy example given above Figure 1, we can say that the policy of Figure 2 is an obligation policy that can efficiently be modeled as a R2ML derivation rule. Providing a set of credentials by a party may satisfy the sequence of constraints in the condition part of a derivation rule, and consequently the privileges in the head of the rule will be given to the party.

To give a clear understanding of how the policy rules can be exchanged by using R2ML, we provide some mappings from the KAoS policy language to R2ML and vice versa.

As shown in Figure 1, one of the parties in the eCommerce transaction negotiation process is using KAoS to define its policies. To exchange such policies with the other side of the negotiation, we should convert them to equivalent R2ML definitions that are then convertible to F-Logic. The KAoS policy of Section 2 is a slightly-modified version of the policy rule which can be stated as: if a buyer provides the delivery information (address, postal code, city, and country) then s/he must provide her/his credit card as well (which means that s/he is obliged to provide the credit card information). It is obvious in the new definition of the rule that its semantic meaning remains untouched, however, it gives us the possibility to define the rule of Section 2 as an Obligation rule in KAoS similar to what we have provided in Figure 3.

The eCommerce negotiation agent on the server side (SellerAgent) converts its policy (rule) from F-Logic to R2ML and sends it to the BuyerAgent (steps 5 and 6 in Figure 1). This transformation process between the F-logic rule and the equivalent R2ML rule is based on mappings shown in Figure 8. Since BuyerAgent uses KAoS to represent policies, it needs to translate the received policy in the R2ML interchange format to the KAoS format. This translation is based on conceptual mappings between KAoS and R2ML, described in the rest of this section.

In Section 3.1, we have mentioned that a KAoS policy is an object of the Policy class in KPO [20] with its attributes instantiated to a set of users, events, and resources that make the policy fire. Considering the KAoS policy element (e.g., Figure 3) as a rule, the action referred to from the oblige element is enforced upon the occurrence of the events described in the requiresConditions element. In addition, some constraints can be placed on the definition of the ObligeAction through using OWL restrictions. Thus, to model the KAoS policy with a derivation rule, we place the decisive content of the oblige element (i.e., the corresponding action, context, and priority) in the conclusion part and the content of the requiresConditions element in the condition part of the rule. An obliged action consists of the action to be performed, its actor, and the context of performing the action with some of them having a restrictive meaning that should also be placed in the condition part of a derivation rule. The result of evaluating a policy rule in KAoS is an object instantiated from a KAoS policy class in KPO that associates the policy object to the context and the action that is obliged to be performed. An R2ML ObjectDescriptionAtom seems to be the appropriate element to define the obligation policy element. R2ML is agnostic about the classes that its objects can be instantiated from. Thus, an R2ML object can be instantiated from an OWL class which has been defined somewhere on the Web. As a result, an R2ML ObjectDescriptionAtom can have a KAoS policy class as its base type and can represent the properties of this policy object as property/term pairs that also refer to the classes in KPO.

To model the condition part of a KAoS policy, we employed R2ML ReferencePropertyAtoms. Similar to the control part, conditions in KAoS are instances of the classes that are defined over an occurred action or state with a set of properties that restrict the action, its actor, or its context. Although the conditions of a policy rule could also be modeled with ObjectDescriptionAtom, the main reason for choosing ReferencePropertyAtom was to be compliant with other R2ML transformations (e.g., transformations between F-Logic and R2ML from the previous section). It simplifies the later conversions of the policies to other rule languages for which we have R2ML transformations already defined (e.g., F-Logic). Moreover, a ReferencePropertyAtom triple models a binary predicate. A set of ReferencePropertyAtoms with the same subject element can always be combined and converted to any element of
A KAoS policy might also have a trigger element. This element is only used with NegObligation- and PosObligation-Policies showing a set of events that trigger the occurrence of an action. In our transformations, we deal with those elements as ReferencePropertyAtoms in the condition part as well. However, to discriminate them from the preconditions, we annotate them as triggering elements.

We show an excerpt of transformation rules between KAoS and R2ML in Figure 9. The XSLT implementations of our transformations are available in [23], where further details can be found.

KAoS uses role-value-map technique to deal with dynamic allocation of values to variables. However, in our implementation, we use a simpler model of defining variables and convert role-value-mapped elements of KAoS to a variable-like definition using R2ML ObjectVariable. This makes the process of converting R2ML variables to the variables of other languages easier. As mentioned before, we map the class definitions for OWL to ObjectClassificationAtoms in R2ML and the objects, which this R2ML element classifies, can be variables representing the completely extensional set for the class, or they can be single objects referring to one single entity in the target domain. To model role-value-maps of KAoS, the class that entails the set of possible values is modeled as an ObjectClassificationAtom with a variable assigned to it. However, the set of values for this variable is restricted to only those instances of the class that have explicitly been defined in the definition of the policy.

Figure 10 shows an excerpt of the policy that we described in Figure 3, converted to its R2ML equivalent based on the transformation rules explained in Figure 9.

Along with mappings and transformations from KAoS to R2ML, we have also defined transformations in opposite direction, from R2ML to KAoS. Transforming back to KAoS is a much simpler task now that we have the transformation rules defined. Of course, the mappings are bidirectional (as for R2ML to F-logic). One of the main issues to define the appropriate transformations from R2ML elements to OWL classes is how to trans-
form the ReferencePropertyAtoms of an R2ML derivation rule with a lower arity to an OWL (or KAoS) class with a higher arity. In our generated R2ML excerpts, each object is associated with a class through using ObjectClassificationAtom, and for each object (as the subject), possibly there is a set of ReferencePropertyAtoms defining restrictions over its properties. When transforming from R2ML back to OWL, an OWL (or KAoS) class can be generated by finding, for each ObjectClassificationAtom, the entity (i.e., the instance or the variable) that it defines, the class that this entity is instantiated from, and the set of ReferencePropertyAtoms that have this entity as their subject. This way we can move from atoms of lower arity in R2ML to classes of higher arity in OWL or KAoS.

Besides this transformation between R2ML and KAoS, we have also defined mappings and XSLT-based transformations between R2ML the Rei policy language [9] and working on the mappings between the WS-Policy and XACML languages and R2ML.

7. Discussion

According to the example of Section 2, collaborating companies with broker agents and services similar to BuyerAgent and SellerAgent would make use of R2ML in many ways. The rules expressing the policies for different business systems and services (e.g., broker agents acting on behalf of buyers or web services representing the facilities of an online store) could be expressed in different rule languages (in this case KAoS and F-logic), but still work with other systems by employing R2ML as an interchange format. On the other side, assuming that BuyerAgent and SellerAgent are products of different companies using different internal policy (rule) languages, it would still be possible for them to work in real-time. When these two systems

![Figure 10. An excerpt of mappings of KAoS elements from Figure 3 a) to R2ML b)](image-url)
need to exchange policy information of their clients they would use R2ML to enable the interchange in real-time. When SellerAgent sends its initial policy information to BuyerAgent, SellerAgent uses R2ML. BuyerAgent takes that policy and translates it from R2ML to its internal representation (KAoS) in order to determine what it needs to do in the process of negotiation.

Besides integration between different policy and rule languages, as shown in this paper, it is also possible to map the descriptions of a semantic web service written in WSDL-S or OWL-S to and from R2ML [9]. This solution in designing an interchange format for Semantic Web services is completely independent from the architecture adapted for service discovery. Availability of transformations from each policy-aware Semantic Web service description to R2ML and back enables the brokers to surf the Web for the desired services regardless of the language they have been originally described in [12].

Semantic Web service descriptions are the most recent solution to extend the interoperability between broker agents and service providers. A description for a Web service is most likely the main source of exchanging information about what can be used in a service, how it can be used, and under which circumstances. Further to describing a service through a series of known terms (i.e., indicating what the service is) and the set of inputs, outputs, functions, and procedures (i.e., how it behaves), the high level business policies and constraints for using a service (circumstances) are also placed in its description. The current proposals for Semantic Web service languages leave the definition of the business rules to the domain specific rule languages (e.g., WSML-Rules in WSMO and RuleML, Rei, or SWRL in OWL-S). To make the current proposals highly interoperable, not only the descriptions for web services should be exchanged, but also the policies should be shared. Our proposal for using R2ML to exchange policies can be considered as a step towards making this interoperability possible.

However, exchanging the policies is more critical than exchanging other business rules as the policies control the behavior of the system and problems in policy transformations may lead to noncompensatable information loss and security threats. We are planning to conduct detailed research to extract the concepts that will be missing while transforming from one policy language to another. Our early research in this area shows that R2ML can cover the concepts for the policy languages, but it depends on the target policy or rule language to be flexible enough to cover the concepts of the source language. From KAoS to F-logic, as F-Logic is a general rule language, the conversion of the concepts is fully possible. So, all KAoS rules can be expressed in F-Logic (although we should consider the differences between the underlying logics, i.e., description logic versus computational logic). Nevertheless, it does not mean that from F-Logic to PeerTrust we will have similar constructs for the policy rules. It is important to analyze how these concepts can be transformed to PeerTrust. In [10], we have investigated the similarities between Rei and KAoS and showed that Rei and KAoS are close enough to cover most of the concepts for the general policy rules, i.e., permission, prohibition, obligation, and dispensation. Doing a similar detailed comparison will be conducted for PeerTrust so that we can eventually cover the concepts between these policy languages as well. It will enable us to move towards a general policy interchange framework that will support various policy languages. Our initial research also demonstrates that we can translate KAoS policies into other rule languages that support derivation rules for which we already have translators to/from R2ML (e.g., JBoss, Jess, and Jena). Transforming between R2ML and any of the languages listed above helps in the sense that we can exploit the use of the reasoners that have been developed for these languages. For example, Jess, has a reasoner that can work over the rules defined for it. This enables us to transform our policies (defined in e.g. KAoS) to Jess rules and integrate them in a system that works with the Jess rules and reasoner. This in turn provides a consistent representation of the knowledge for the system and adds to the interoperability of our Rule language. Nevertheless, the concepts used in our transformations might not be compatible with the concepts available in the target rule language although they are part of the R2ML derivation rule. As an intuitive example, during our transformations from KAoS driven R2ML policies to Jena, the transformator reported an impossibility of converting R2ML qf.Disjunction to an identical concept in Jena. This introduces problems that we may face while transforming concepts such as unionOf from OWL (or KAoS) to Jena. A detailed investigation of this area is subject to the future research.

8. Conclusion

In this paper, we have shown how a comprehensive and general purpose rule markup language such as R2ML can be used as an intermediary language to transform different business rules and policies from one language to another in the process of an eCommerce negotiation between different parties. It was shown how an F-logic rule could be mapped to and from R2ML. It was also illustrated how a derived R2ML rule
from a KaoS policy rule looks similar to a derived R2ML rule from F-Logic and how the conversion from one language to another can help with interchanging their concepts.

Policy-RuleML [17] is a similar attempt in the area of policy transformation. It aims at making RuleML a semantic interoperation vehicle for heterogeneous policies and protocols on the Web. However, to the best of our knowledge there is no proposed work done by this group, and thus our solution seems to be the first practical attempt in the area. The future goal of the research will be combining the semantic web service description languages with policy languages and trying to get different web services with different descriptions and policy languages to work together. The current proposals on combining semantic web services with policy languages have been proposed in [16] that combines WSMO and PeertTrust, [20] that uses KaoS to protect web services, and [7] that combines Rei and OWL-S. Our goal will be to let all of these services to communicate regardless of the languages they use for defining and describing their services and policies. Furthermore, we have developed transformations from OCL to R2ML [14] and vice versa. Another goal will be to make all the transformations between policy languages and OCL consistent, so that we can eventually integrate Semantic Web policies for services with Model-Driven software development approaches. Developing a general policy language based on R2ML to cover the concepts of all abovementioned policy languages is also another future goal in our research.

References